Trillion Dollar Security Project

Security Challenges Overview

ethereum
foundation

Table of Contents

Introduction

Ethereum is the most secure, resilient, and trusted blockchain ecosystem. Over the last 10 years the Ethereum ecosystem has developed the technology, standards, and knowledge that today support an ecosystem used by millions and that is home to more than $600 billion in capital.

But for Ethereum to succeed in the next phase of global adoption, there are still many improvements that must be made. To achieve our community's ambitions, Ethereum must grow into an ecosystem where:

- Billions of individuals are each comfortable holding more than $1000 onchain, collectively amounting to trillions of dollars secured on Ethereum.

- Companies, institutions, and governments are comfortable storing more than 1 trillion dollars of value inside a single contract or application, and are comfortable transacting in comparable amounts.

The Trillion Dollar Security (1TS) project is an ecosystem-wide effort to upgrade Ethereum's security. This report is the first deliverable of the 1TS project. Over the last month, we have gathered feedback from users, developers, security experts, and institutions about where they see the biggest challenges and areas for improvement. Thank you to the hundreds of people and dozens of organizations who have taken the time to share your insights with us.

This report summarizes our findings, covering 6 distinct areas:

1. **User experience (UX)** - Issues that affect users' ability to securely manage private keys, interact with onchain applications, and sign transactions.

2. **Smart contract security** - The security of the smart contract components of Ethereum applications, and the lifecycle of software production that shapes them.

3. **Infrastructure & cloud security** - Issues with the infrastructure (both crypto-specific and legacy) that Ethereum apps depend on, like L2 chains, RPCs, cloud hosting services, and more.

4. **Consensus protocol** - The security properties of the core protocol, which secures the Ethereum blockchain itself from attack or manipulation.

5. **Incident response & mitigation** - The challenges users and organizations face when responding to security breaches, particularly in recovering funds or managing the aftermath.

6. **Social and governance layer** - Ethereum's open source governance, community, and ecosystem of organizations.

This first report is focused on identifying and mapping the problems and challenges that remain. The next step will be to choose the highest priority issues, identify solutions, and work with the ecosystem to address them.

Because the Ethereum ecosystem is decentralized, securing Ethereum is not something that can be done by a single entity. Ethereum's technology stack is built and maintained by independent organizations all around the world, ranging from wallets to infrastructure to developer tooling. While the 1TS project is coordinated by the Ethereum Foundation, we need your help to secure Ethereum.

You can contribute to the 1TS security project by sharing your feedback and ideas:

- Are there problems you see in Ethereum security not included in this report?
- What do you believe are the highest priorities of the issues surveyed below?
- What ideas or solutions do you have for how to address these problems?

We're eager to hear from you at **trilliondollarsecurity@ethereum.org**.

1. User experience (UX)

Security begins with the interface people use to interact with Ethereum. This boundary between users and the blockchain itself is a consistent source of security challenges.

One defining feature of blockchains is the atomic nature of transactions: once an update is recorded into the blockchain, there is no opportunity for intervention or reversal. This provides strong guarantees of consistency and protocol level security, but exposes users to heightened operational risk: a single mistake, compromised key, or rushed approval can lead to irreversible loss.

As a result, a significant burden of security falls on the user. To use Ethereum safely, individuals and organizations must securely hold and manage keys, interact with onchain applications, and use their keys to sign transactions to transfer assets or otherwise update Ethereum's state.

Each of these requirements introduces risks like key compromise or loss, rushed or uninformed approvals, or compromise of the wallet software users rely on to inform and guide them through interacting with Ethereum.

UX security and safety was the top issue identified through feedback and consultation with the ecosystem. These issues affect users of all kinds, ranging from individuals to large enterprises. While many other areas of Ethereum's technology stack are immensely secure, today user experience is the weakest link.

1.1 Key management

Many users are not equipped to safely manage cryptographic keys.

Most widely used software wallets rely on users securely storing seed phrases representing their underlying cryptographic private key, which often leads them to use insecure workarounds like storing seed phrases in plaintext, on cloud services, or writing them down on paper.

Hardware wallets are an alternative, which enable users to manage a cryptographic key stored within a special purpose physical device. However, hardware wallets have their own flaws and attack surface. Hardware wallets can be lost, damaged, or stolen. Many hardware wallets are not open source and may have opaque supply chains, raising the risk of a supply chain attack where compromised devices are sold into the market.

Whether keys are managed in a software or hardware wallet, many users are understandably nervous about self custody when it can be compromised through physical theft or assault.

Enterprise and institutional users face additional challenges in key management. If individual employees hold keys (e.g. as part of a multisig wallet), the organization must be able to replace them and create new ones due to personnel changes over time. Compliance requirements in different industries and jurisdictions may require custom workflows or audit trails not supported by existing wallet software. In some cases, enterprise users turn to third-party custodians for digital assets, which may introduce another layer of security risks to consider.

1.2 Blind signing & transaction uncertainty

Users routinely approve transactions “blindly” without understanding what they are doing. Wallets often present raw hexadecimal data, truncated contract address, or other information that is not sufficient for the user to understand the consequences of a given transaction. This leaves users of all kinds vulnerable to malicious smart contracts, phishing, scams, spoofed interfaces, front-end compromises, and basic user errors.

1.3 Approval and permission management

In many Ethereum applications, it is common for users to grant certain permissions to the underlying application as part of normal use. For instance, a user might grant a decentralized exchange like Uniswap permission to move their tokens in order to swap them for ETH.

These approvals can have limits on the amount, but many wallets default to granting unlimited approvals with no expiry date. There is no way for users to manage or review their outstanding approvals from within most wallets.

This can expose users to malicious apps or compromised frontends, because the default pattern for many users is to grant unlimited approvals which can be used to drain their funds. Even if a user grants an approval to a legitimate smart contract, if that contract were later compromised while the approval remains in place, then the compromised contract could drain the user’s funds.

This is equally a risk for organizational users. For instance, an organization might choose to grant a DEX router unlimited USDC allowance for operational convenience, which then exposes them to risks if the router contract is upgraded.

1.4 Compromised web interfaces

Most users do not directly interact with a smart contract, but rather through a web interface via their mobile device or web browser.

These frontends can be vulnerable to attack through familiar means like DNS hijacking, malicious javascript injection, insecure hosting, or various third party dependencies. A compromised app UX can redirect users of all kinds to malicious smart contracts or lead them to sign misleading transactions.

1.5 Privacy

Privacy can mitigate or magnify security risks for users of all kinds.

Weaker privacy protections expose individual users to a variety of targeted threats like phishing, exploitation, scams, or physical attacks. Many common UX patterns expose users, e.g., address reuse, KYC data, and other metadata leaks.

For institutions and enterprises, privacy is often a fundamental business requirement for compliance reasons or certain use cases. In addition to those issues, it can create exposure to specific security risks. For instance, a user of a supply chain system built on Ethereum may require strong privacy guarantees to protect intellectual property assets that could be compromised if the system was transparent.

1.6 Fragmentation

There is a lack of consistency in how different wallets handle core behaviours like displaying transactions, handling approvals, or labelling contracts. This fragmentation of user experience adds friction to the user's ability to learn how to safely use wallets, and increases risks.

For instance, users cannot rely on consistent UX cues to protect themselves from phishing and spoofing as they differ across wallets. Users cannot form reliable expectations about how Ethereum works if each tool functions differently.

2. Smart contract security

Smart contracts are the onchain components of Ethereum applications: the code that holds funds, defines access controls, and enforces the application's business logic. Because smart contracts are typically transparent and accessible to anyone, they are a critical attack surface when considering security in the Ethereum ecosystem.

Smart contract security has radically improved over Ethereum's history. Early security incidents like the DAO hack motivated the ecosystem to professionalize and improve safeguards across the software lifecycle that leads to code being deployed onchain. Key advancements include:

- Security auditing became a standard practice, with several security firms entering the ecosystem and developing expertise.
- Tooling, testing, and static analysis systems matured and became standard practice.
- Libraries of pre-audited common components gave developers secure-by-default building blocks.
- Formal verification techniques were adopted, especially for bridges, staking systems, and high value contracts.
- The ecosystem's security culture and best practices improved.
- The creation of significant bounty programs that hardened the app layer.

However, there remain weaknesses and areas for improvement in this domain.

2.1 Contract vulnerabilities

Despite advances in smart contract security, there are still vulnerabilities that can lead to significant security issues, including:

- **Contract upgrade risk**. Some contracts are designed to be modifiable after deployment, to enable a development team to continue to update and improve an application. However, this introduces risks. Upgrades could result in new vulnerabilities, or total loss of user funds in the case of a malicious upgrade.

- **Re-entrancy**, where contract A calls an external contract B before updating its own internal state, and contract B calls back to the original contract A before the first call finishes.

- **Unsafe use of external libraries**, where a contract calls an external library that may be unaudited, malicious, or upgradeable.

- **Unaudited components**. While auditing and use of standard libraries has improved, developers sometimes rely on unaudited components in their applications.

- **Access control failures**, where permissions are misconfigured or defined too broadly, allowing attackers to take malicious actions.

- **Unauthorized Access**, where a private key that is able to control the contract is obtained by a malicious actor.

- **Bridges and crosschain interactions**. Bridges and crosschain protocols introduce additional complexity, and attackers can exploit weaknesses in how crosschain messages are passed or validated.

- **Externally Owned Account (EOA) delegation or signature misuse**. Malicious applications may trick users into signing over full delegation of their account to another party, enabling theft. Malicious applications can also use signed messages from the user in unexpected ways, e.g., in a replay attack.

- **Emerging risk of bugs introduced by AI code generation or automated refactoring tools.**

2.2 Developer experience, tooling and programming languages

Vulnerabilities end up in deployed code as a result of developer error. Improved developer tooling has made it significantly easier to deploy safe smart contracts. However, issues remain.

- **Lack of secure defaults in popular frameworks**. Some tools prioritize flexibility or speed over safety, setting insecure defaults like unlimited token approvals in the approve() function, or failing to include access control patterns by default.

- **Custom code for advanced operational controls**. Institutional users with complex operational requirements often must build required features from scratch, increasing the risk of vulnerabilities. There is a lack of standardized secure components or frameworks for advanced security workflows.

- **Inconsistent testing coverage** across tooling stacks, as well as a lack of norms around using proven techniques like fuzzing or invariant checking.

- **Low adoption of formal verification methods**. Formal verification techniques are powerful, but they are complex, costly, require specialized domain expertise, and are not well integrated into standard developer workflows, where they could be used much earlier in the production of software to verify safety at the specification stage.

- **Issues related to contract verification**. Users and developers cannot easily assess the trustworthiness of deployed contracts, the extent of their security validation (e.g. code audits), or the presence of latent risks. While solutions exist for this purpose, many issues remain. Tooling that addresses these issues is not widely adopted, the standards that would unify the approaches remain fragmented, and some of the existing services are themselves centralized dependencies.

- **Compiler risks**. Compilers (the software that converts human readable code like Solidity into the bytecode used by the EVM itself) can have flaws which introduce errors into smart contracts before they are deployed. The Ethereum ecosystem today mostly depends on the solc compiler, meaning a bug could have widespread effects.

- **Programming language diversity and depth**. While Solidity has a deep tooling ecosystem built on it, some developers want more modern safety features found in other programming languages, like memory safety.

2.3 Risk assessment of onchain code

Institutions and enterprises have existing processes, standards, and requirements for evaluating the security of technology and systems that they depend on. However, existing frameworks often do not cleanly map onto smart contracts, usually assuming mutable code, centralized change control, and clear lines of accountability or legal liability. Systems built on smart contracts may sometimes break those assumptions, making it difficult for organizations to adopt Ethereum and manage risk appropriately.

3. Infrastructure and cloud security

Many uses of Ethereum depend on a variety of infrastructure providers, including both crypto-specific infrastructure (e.g., Layer 2 chains, RPC providers) and traditional cloud and internet infra (e.g., AWS, CDN, DNS).

These systems are an attack surface both for the wallet and application layer (e.g., RPC endpoints for wallets) and for the Ethereum protocol itself (e.g., many validators are hosted on cloud infrastructure). Private key compromise, phishing, and lack of granular access controls can lead to large-scale outages, theft, or unauthorized changes, even if the underlying blockchain protocol remains secure.

3.1 Layer 2 chains

Layer 2 chains (L2s) serve as extensions for Ethereum, enabling faster and lower fee environments while retaining some of the characteristic security guarantees of Ethereum mainnet (depending on their specific design). However, they also have their own distinct attack surfaces including:

- **Multi-hop bridged asset complexity**. When assets travel between L1 and multiple L2s, they are exposed to multiple sets of contracts all of which must be secure. Mismatched accounting or outages in L2 chains can introduce vulnerabilities that can be exploited by attackers.

- **Rollup L2s rely on proving systems to enforce correctness of state updates.** Bugs or misconfigurations in these systems can stall or prevent finalization, or allow finalization of false state updates leading to loss of user funds.

- **Security councils are groups of keyholders who serve as a "backup" mechanism to upgrade L2 software or respond to certain emergencies.** Security councils themselves pose risks, as compromise or collusion among members could put user funds at risk or freeze assets.

See L2Beat for a detailed framework and monitoring dashboard that evaluates and compares L2 performance and security.

3.2 RPC and node infrastructure

Ethereum applications depend on a small number of infra providers for RPC access, APIs and node services. This includes crypto-specific infra providers, as well as traditional cloud services that are commonly used to host nodes (e.g., AWS, Cloudflare, Hetzner).

If these infra providers went offline or attempted to censor or throttle access, many users could be prevented from accessing Ethereum through their wallet or application, until they were able to migrate to a new RPC or other infra provider. Some of these providers have previously suspended or shut down accounts associated with blockchain activity, raising concerns about their long-term reliability for decentralized applications.

3.3 DNS level vulnerabilities

The Domain Name System (DNS) is a foundational layer of the internet, but it is also centralized and can be compromised. Many users access apps through web domains, which are susceptible to:

- DNS hijacking where an attacker inserts a malicious false frontend.
- Domain seizure, where a government or registrar can seize domains.
- Phishing via lookalike domains, where attackers register near identical names to confuse users.

3.4 Software supply chain and libraries

Ethereum developers rely on open-source libraries, often pulled directly from services like npm, crates.io, or GitHub. If these libraries are compromised, they can be a vector for attacks like:

- **Malicious package injection**, where attackers compromise a widely used package or publish one under a similar name.
- **Hijacked dependencies**, where maintainers lose control of a project and a malicious actor introduces harmful code.
- **Developer compromise**, where packages installed contain code that gives the attacker control over the developer's computer.

3.5 Frontend delivery services and related risks

Many Ethereum applications serve their frontends via a Content Delivery Network (CDN) or cloud-based hosting platform (e.g., Vercel, Netlify, Cloudflare). If these services are compromised, they can be a vector for attacks like malicious javascript injection, where attackers serve an altered frontend to users.

3.6 Internet Service Provider level censorship

Internet Service Providers (ISPs) or nation states can use control of underlying internet infrastructure to censor access to Ethereum. For example, these attacks could include:

- Blocking or throttling traffic to common Ethereum ports.
- Filtering DNS requests that resolve to Ethereum related services.
- Geofencing or IP bans against known Ethereum nodes.
- Deep packet inspection to identify and censor Ethereum protocol related traffic.

Many of these basic techniques are already used by authoritarian governments around the world to suppress access to information, protest tools, or cryptocurrencies today.

4. Consensus protocol

Ethereum's consensus protocol defines how the network updates the state of the Ethereum blockchain and comes to agreement. This protocol is at the foundation of what makes Ethereum a trustworthy platform for money, finance, identity, governance, real world assets, and more.

Ethereum's consensus protocol has proven robust in practice, with zero downtime since first launching in 2015 and across several upgrades. However, there remain long-term areas for improvement to make the system more resilient and secure.

4.1 Consensus brittleness and recovery risks

Ethereum's fork choice and finality rules are resilient, but they are not invulnerable. During certain edge case conditions (like prolonged validator disagreement, client bugs, or network partitions) consensus could stall or temporarily diverge. In extreme conditions, this could lead to cascading validator penalties through inactivity leaks or slashing, which could further lead to capital flight from validators.

4.2 Client diversity

Ethereum's industry-leading client diversity protects the network from bugs in any single client. However, client diversity could still be improved with more adoption of minority clients to reduce these risks even further.

4.3 Staking centralization and pool dominance

A significant amount of validator weight is concentrated in liquid staking protocols, custodial services, and large node operators. This concentration can lead to risks like:

- **Governance capture or influence**. If entities controlling large amounts of stake (or entities with legal power to influence those entities) coordinated together, they could have outsized influence on which blocks are proposed and attested, potentially censoring users, or influencing protocol upgrades.
- **Homogeneity in client choice and infrastructure setup**, which can increase correlated failure risks.

4.4 Undefined social slashing and coordination gaps

In some extreme failure modes, Ethereum would rely on "social slashing" to penalize validators who acted maliciously to attack the network (see section 6.1). However, the infrastructure, norms, and expected

processes for this kind of slashing are underdeveloped. There is no established mechanism that the community would use to engage in this process.

4.5 Economic and game-theoretic attack vectors

Many potential economic attack vectors remain understudied, including:

- **Griefing attacks or slash griefing**. Validators may incur costs or slashing penalties not due to their own faults but because of adversarial behavior intended solely to harm others at a net cost to the attacker.
- **Strategic exits or timed inactivity**. Validators could intentionally go offline or exit at critical times to maximize profits or disrupt consensus with minimal penalties.
- **Collusion among validators or relays**. Coordinated behavior between validators or between relays and validators could reduce decentralization, or extract MEV.
- **Exploitation of edge-case incentives in MEV, proposer-builder separation, or liquid staking design**. Actors may manipulate rare protocol conditions to gain outsized rewards.

4.6 Quantum risk

Ethereum's core cryptography (e.g. elliptic curve signatures like secp256k1) could one day be broken by quantum computers. While this is not an imminent risk, a credible threat could instantly render existing wallets, contracts, and staking keys vulnerable. This future challenge weakens Ethereum's long-term guarantees to users.

Migration paths to quantum-resistant cryptography (e.g., via post-quantum signature schemes) need to be designed, tested, and possibly embedded in the protocol years before they are needed. Organizations across the Ethereum ecosystem, including the Ethereum Foundation, are actively exploring these options and monitoring risks.

5. Monitoring, incident response, and mitigation

Even an idealized blockchain ecosystem will have risks, attacks, and vulnerabilities. When things do go wrong, there must be effective systems to mitigate, detect, and respond. Challenges here include:

- **Reaching the affected team**. It can be difficult to get in contact with the team whose application has been compromised. This can lead to hours of delays, limiting responders' ability to recover funds.
- **Escalating issues at related organizations**. When the issue involves a platform (like a social network or centralized exchange) it can be challenging for responders to escalate the issue if they do not have a pre-existing contact.
- **Response coordination**. It is often unclear how many incident response teams are assisting the affected application, leading to miscommunication or wasted effort when a group effort might have been more effective.
- **Lack of monitoring capabilities**. It can be difficult to monitor for onchain and offchain issues, which would provide early warning and ensure a swift response to threats.
- **Access to insurance**. Insurance is an essential tool for mitigating losses in most traditional systems that deal with money, financial systems, identity, and other valuable information. However, today few insurance options are available from traditional financial services for the crypto ecosystem.

6. Social layer and governance

Ethereum's "social layer" refers to the set of people, organizations, companies, governance processes, and cultural norms that influence how the Ethereum ecosystem behaves. This social layer is itself vulnerable to certain attacks or risks, which can then influence the security and reliability of Ethereum.

These risks tend to be more long-term oriented, and concern Ethereum as a whole rather than the security of individual users or applications.

6.1 Stake centralization

Centralization of large amounts of stake can pose risks to Ethereum as a whole if the entities controlling that stake decide to collude.

This economic centralization creates the potential for social governance capture. If a small group of validators controls a supermajority of stake, they could:

- Coordinate on or resist forks.
- Censor certain transactions or contracts.
- Undermine community consensus by threatening exit or opposition.

If this extreme scenario were to happen, the Ethereum community has suggested that "social slashing" might be the answer. Social slashing is the use of offchain social consensus to decide to slash misbehaving validators, as a check on their power. But no clear norms, procedures, or tooling exist to enact such measures (see section 4.4).

6.2 Offchain asset centralization

Ethereum hosts significant amounts of real world assets, where the assets are held offchain in bank accounts or other deposits, which are then traded onchain via tokens that represent a claim on the offchain assets. For instance, many large stablecoins function this way.

The institutions that hold the offchain deposits may have influence over the Ethereum ecosystem. For instance, during an extreme scenario where there is a contentious fork or network upgrade, large depositors can influence which chain becomes widely accepted by only choosing to recognize tokens on one chain or the other.

6.3 Regulatory attack or pressure

Governments and regulators could pressure various entities that control important components of the

Ethereum stack to censor or otherwise interfere with the Ethereum protocol. Institutional users of Ethereum could also be impacted by these pressures, which would have further consequences for their users (e.g. a bank that can no longer offer certain crypto products due to regulatory bans).

6.4 Organizational capture of governance

Ethereum's open source governance and development processes are driven by a diverse and global set of teams and companies that maintain core client software, infrastructure, and tooling.

Various forms of influence (corporate acquisitions, funding dependencies, employment of key contributors, conflicts of interest inside existing orgs) could gradually shift the culture and priorities of Ethereum governance. This may lead to alignment with specific commercial or external interests that diverge from the community-driven ethos and established roadmap, potentially weakening Ethereum's neutrality and resilience over time.